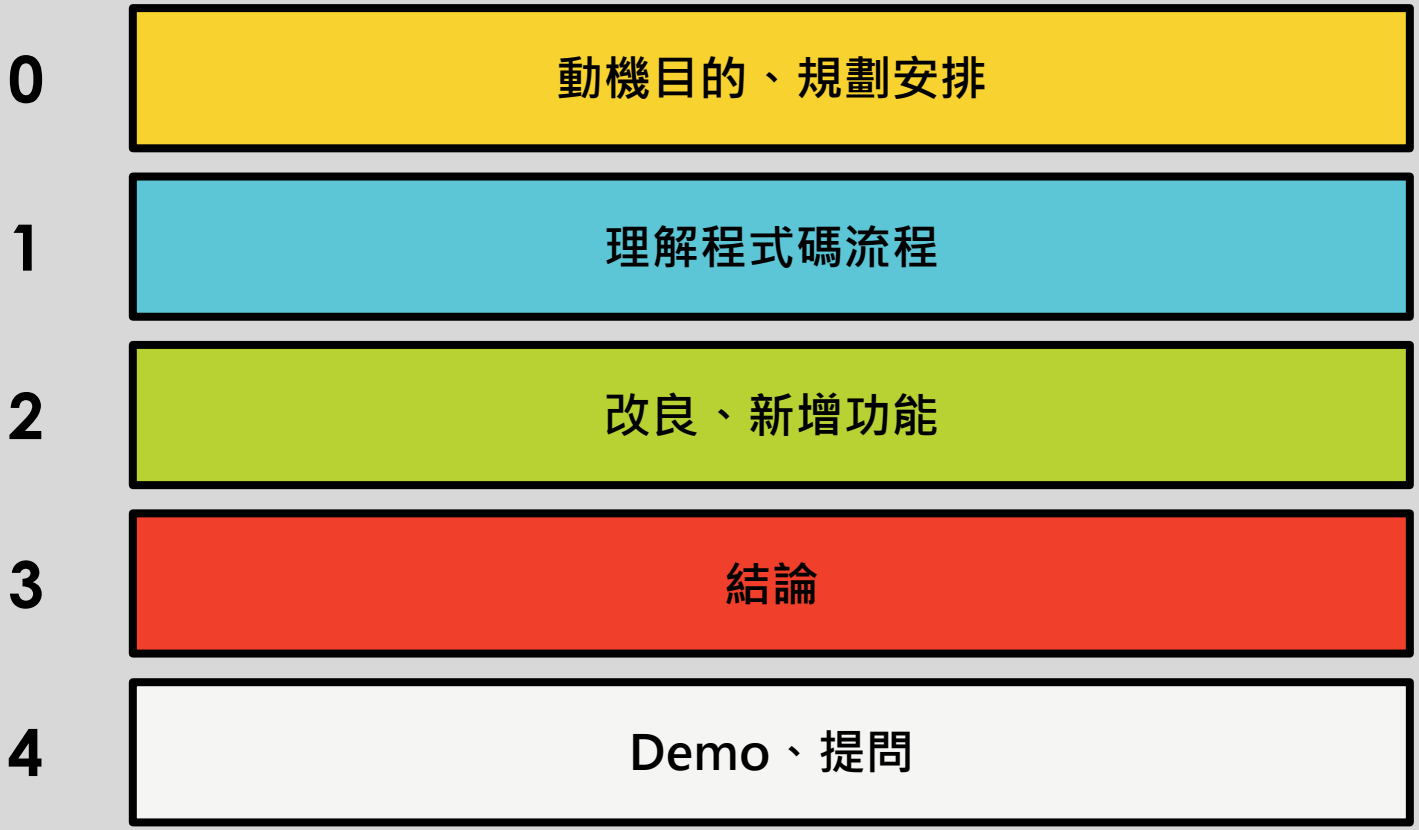
QT & VTK期末報告

41041223 41041217 41041220 41041222
黃柏浚　　李祐賢　　徐唯博　　許睿宏

Contents



Toolkit

VTK 8.2.0
SOURCE版本

CMAKE LATEST
RELEASE版本 (3.24.2)

QT 5.9.4

VISUAL STUDIO 2022

踩過的坑

安裝步驟一樣，但是電腦不同安裝環境失敗

VTK真的比較簡單嗎？

Three.js真的比較難嗎？

VTK9.2 編譯出錯...

官網的範例是VTK9的，VTK8有的無法使用

網路上教學很少...

功能難產...

找不到偵錯的方式

C++與QML之間的連接有點難

Debug模式下可以執行，Release不行...

找不到qDebug()輸出的內容

網路上的教學，版本都不同，很多教學已經過時了...

CMake中隱藏了輸出控制台...

Qt信號與槽的機制...

環境的不同會造成前期安裝環境會失敗

出現錯誤但沒有錯誤訊息

VS2022會有未知錯誤

QML傳輸到C++很簡單，但C++到QML很複雜

QtVtk程式碼有點難

單純VTK可以做到，但是加上Qt卻失敗了...

我感覺老師跟助教好像也不會 :(

動機目的

動機：

鑽研和理解程式碼在做什麼，並且能夠在原程式碼的基礎上進行改良及新增功能等，製作一款專屬於我們這組的一個QtVtk視覺化工具。

目的：

學習VTK視覺化工具函數庫、Qt類別、QML語法、資料結構演算法及C++繼承多型指標重載的特性

規劃安排

學習規劃：

在新增功能以前，我們必須先讀懂原程式碼在做什麼，因此在學習規劃上面第一步是先讀懂程式碼。小組互相分工，分配每個人閱讀的部分並寫上註解，完成後互相觀看彼此的註解，試圖理解這份程式碼每一個區塊的作用。第二步會與組員一同討論，並基於原程式碼進行改良甚至是新增功能。最後一步，組員之間互相幫忙，將學到的知識傳授給理解較慢的人。

小組分工：

41041223 (組長) 負責 規劃、安排及整合程式碼

41041217 (組員) 負責 CanvasHandler、CommandModelAdd、CommandModelTranslate

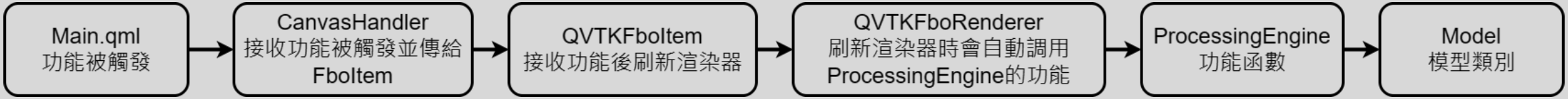
41041220 (組員) 負責 Model、ProcessingEngine

41041222 (組員) 負責 QVTKFboItem、QVTKFboRenderer

QtVtk程式碼的框架結構

- Main.qml : 程式的介面設計
- Main.cpp : 主程式入口
- CanvasHandler : 距離QML最近， 可以直接與QML進行對話，功能回調函數入口
- QVTKFboItem : 接收到功能後刷新渲染器
- QVTKFboRenderer : 介面、互動渲染器，滑鼠事件重載，模型Pick事件，刷新渲染器會自動調用ProcessingEngine內的功能
- Model : 模型的類別
- Command開頭的檔案 : 執行不同的任務
- ProcessingEngine : 讀模型檔案及模型功能

過程如下圖所示



VTK的框架結構

- vtkSource : 輸入的數據
- vtkFilter : 數據過濾
- vtkMapper : 將數據轉換成點、線及幾何，負責存放數據和渲染信息（生成映射圖元）
- vtkActor : 場景的設定，負責控制顏色、不透明度等參數（可展示對象）
- vtkRender : 渲染器（渲染場景）

數據源(Source) -> 過濾器(Filter) -> 映射器(Mapper) ->演員(Actor) ->

渲染器(Render) -> 窗口(RenderWindow) -> 窗口交互(Interactor)

過程如下圖所示

41041217

主程式 main

一開始會進入主程式，接著會直接呼叫 CanvasHandler.cpp

```
1   #include "CanvasHandler.h"
2
3   int main(int argc, char **argv)
4   {
5   #ifdef __linux
6       putenv((char *)"MESA_GL_VERSION_OVERRIDE=3.2");
7
8       // Fixes decimal point issue in vtkSTLReader
9       putenv((char *)"LC_NUMERIC=C");
10  #endif //LINUX
11      //程式開始直接呼叫CanvasHandler
12      CanvasHandler(argc, argv);
13
14      return 0;
15  }
16
```

41041217

CanvasHandler

CanvasHandler主要用於建立QML與C++的連線，讓QML調用函數，以及基礎的細項設定。

```
//跟QML建立連線橋梁，這個文件有點像是主文件（Main），QML中的按鈕事件或是其他功能事件，按鈕按下後都會來到這裡調用函數

//程式一開始會從main.cpp直接呼叫這段
CanvasHandler::CanvasHandler(int argc, char **argv)
{
    QApplication app(argc, argv);
    //QApplication負責程式的初始、結束及處理事件的迴圈等，並提供基本的視窗外觀

    QQmlApplicationEngine engine;
    //QQmlApplicationEngine提供從一個QML文件裡加載應用程式的方法

    app.setApplicationName("QtVTK");
    //設定app名稱

    app.setWindowIcon(QIcon(":/resources/bq.ico"));
    //設置圖標
```

CanvasHandler

qmlRegisterType指令是讓在C++編寫好的class可以在QML裡被調用

```
qmlRegisterType<QVTKFramebufferObjectItem>("QtVTK", 1, 0, "VtkFboItem");
// Register QML types : 註冊QML類型
/*qmlRegisterType為連結C++與QML的函式
第一個參數為QML中放在import後的內容
第二、三個參數為版本
第四個參數為QML中的class
例:
import QtVTK 1.0
VtkFboItem {
        id: vtkFboItem
        objectName: "vtkFboItem"
        .....等
}*/
```

main.qml

```
1  import QtQuick 2.9
2  import QtQuick.Controls 2.2
3  import QtQuick.Dialogs 1.2
4  import QtQuick.Window 2.3
5  import QtQuick.Controls.Material 2.2
6  import QtVTK 1.0
7
```

```
VtkFboItem {
    id: vtkFboItem
    objectName: "vtkFboItem"
    anchors.fill: parent
```

CanvasHandler

初始化ProcessingEngine

setContextProperty讓QML認得CanvasHandler

```
m_processingEngine = std::shared_ptr<ProcessingEngine>(new ProcessingEngine());
// Create classes instances
//初始化ProcessingEngine，ProcessingEngine主要功能為讀檔及初始化位置

QQmlContext* ctxt = engine.rootContext();
// Expose C++ classes to QML :將C++的類別暴露給QML
//加這行才可以開始抓東西出來給QML使用

ctxt->setContextProperty("canvasHandler", this);
//setContextProperty( "在QML裡被呼叫時使用的名稱" , 在C++裡的位置 )
//加這行 QML 才會認得 canvasHandler 是誰、要到哪裡調用函數
```

```
        MouseArea {
            acceptedButtons: Qt.LeftButton
            anchors.fill: parent

            onPositionChanged: {
                canvasHandler.mouseMoveEvent(pres
            }
            onPressed: {
                canvasHandler.mousePressEvent(pre
            }
            onReleased: {
                canvasHandler.mouseReleaseEvent(p
            }
        }
    }

    Button {
        id: openFileButton
        text: "Open file"
        highlighted: true
        anchors.right: parent.right
        anchors.bottom: parent.bottom
        anchors.margins: 50
        onClicked: canvasHandler.showFileDialog =

        ToolTip.visible: hovered
        ToolTip.delay: 1000
        ToolTip.text: "Open a 3D model into the c
    }
```

CanvasHandler

各種滑鼠動作事件，QML會來這裡調用函數，這裡的函數會指向QVTKFramebufferObjectItem的函數做後續處理

```
//滑鼠點擊事件
void CanvasHandler::mousePressEvent(const int button, const int screenX, const int screenY) const
{
    qDebug() << "Can

    m_vtkFboItem->se
    //選擇模型
}
```

```
onPositionChanged: {
    canvasHandler.mouseMoveEvent(pressedButtons, mouseX, mouseY);
}
onPressed: {
    canvasHandler.mousePressEvent(pressedButtons, mouseX, mouseY);
}
onReleased: {
    canvasHandler.mouseReleaseEvent(pressedButtons, mouseX, mouseY);
}
```

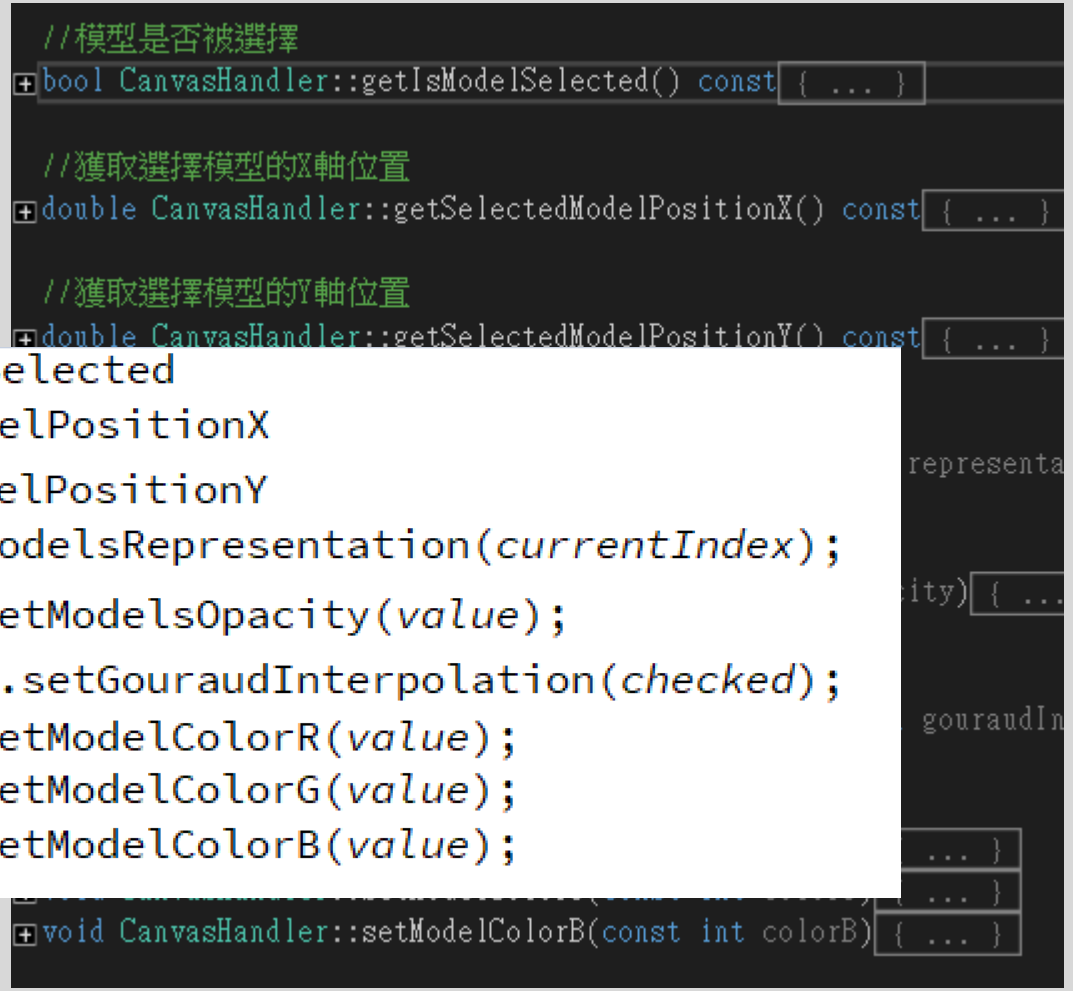
```
void CanvasHandler::mouseReleaseEvent(const int button, const int screenX, const int screenY)
{
    qDebug() << "CanvasHandler::mouseReleaseEvent()";

    if (!m_vtkFboItem->isModelSelected())
    //檢查是否有選擇模型，如果沒有
```

CanvasHandler

右圖皆為函數調用

都指向QVTKFramebufferObjectItem.cpp的函數

```
visible: canvasHandler.isModelSelected
text: "X: " + canvasHandler.modelPositionX
text: "Y: " + canvasHandler.modelPositionY
onActivated: canvasHandler.setModelsRepresentation(currentIndex);
onValueChanged: canvasHandler.setModelsOpacity(value);
onCheckedChanged: canvasHandler.setGouraudInterpolation(checked);
onValueChanged: canvasHandler.setModelColorR(value);
onValueChanged: canvasHandler.setModelColorG(value);
onValueChanged: canvasHandler.setModelColorB(value);
```



CanvasHandler

openModel函數功能就是選擇並檢查檔案格式

addModelFromFile是QVTKFramebufferObjectItem的函數

```
void CanvasHandler::openModel(const QUrl &path) const  //openModel功能:開啟3D圖檔
{
    //path為所選檔案之路徑
    qDebug() << "CanvasHandler::openModel():" << path;

    QUrl localFilePath;
    //QUrl在qurl.h中被定義，功用為提供接口使用URLs(網址)
    //先定義localFilePath備用，待會用來儲存網址

    if (path.isLocalFile())
    //如果所選檔案是本地檔案的話(例:"file:///C:/Users/user/Downloads/2.stl")
    {
        // Remove the "file:///" if present
        localFilePath = path.toLocalFile();
        //toLocalFile()為托放功能，把所選的本地檔案放進localFilePath
    }
    else
    {
        localFilePath = path;
    }//否則直接代入localFilePath

    m_vtkFboItem->addModelFromFile(localFilePath);
    //用指標方法存取檔案來源到m_vtkFboItem
}
```

CommandModelAdd

CommandModelAdd主要是QVTKFramebufferObjectItem跟ProcessingEngine的溝通橋樑

run函數會呼叫ProcessingEngine的addModel跟placeModel函數

execute是執行函數，會呼叫render的函數新增一個Actor

```
void CommandModelAdd::run()
{
    qDebug() << "CommandModelAdd::run()";

    m_model = m_processingEngine->addModel(m_modelPath);
    //add model 返回一個model類型
    m_processingEngine->placeModel(*m_model);
    //設定model在原點

    m_ready = true;

    emit ready();
    //發射信號
}
```

```
void CommandModelAdd::execute()
{
    qDebug() << "CommandModelAdd::execute()";

    //幫model新增一個actor
    m_vtkFboRenderer->addModelActor(m_model);

    emit done();
}
```

41041220

ProcessingEngine - 載入模型

到ProcessingEngine進行讀檔，根據檔案的不同，使用不同的Reader去進行讀檔

```
//讀檔
const std::shared_ptr<Model> &ProcessingEngine::addModel(const QUrl &modelFilePath)
{
    qDebug() << "ProcessingEngine::addModelData()";

    QString modelFilePathExtension = QFileInfo(modelFilePath.toString()).suffix().toLower();
    //OBJ讀檔
    vtkSmartPointer<vtkOBJReader> objReader = vtkSmartPointer<vtkOBJReader>::New();
    //STL讀檔
    vtkSmartPointer<vtkSTLReader> stlReader = vtkSmartPointer<vtkSTLReader>::New();
    vtkSmartPointer<vtkPolyData> inputData;

    if (modelFilePathExtension == "obj")
    {
        // 若是讀取到OBJ檔,就用此方法
        objReader->SetFileName(modelFilePath.toString().toStdString().c_str());
        //更新資料
        objReader->Update();
        inputData = objReader->GetOutput();
    }
    else
    {
        // 若是讀取到STL檔,就用此方法
        stlReader->SetFileName(modelFilePath.toString().toStdString().c_str());
        stlReader->Update();
        inputData = stlReader->GetOutput();
    }
```

41041220

ProcessingEngine - 載入模型

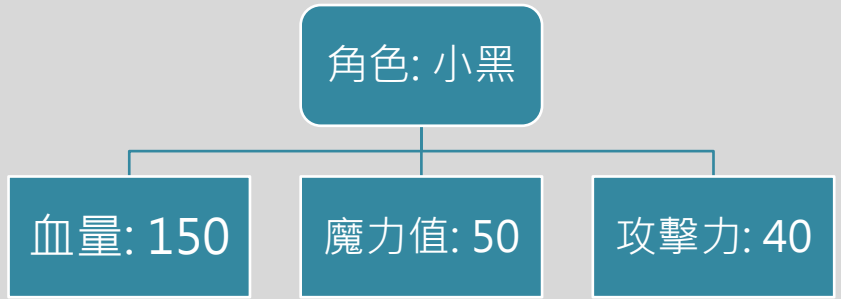
接著將讀完後的檔案數據丟入處理Polydata的函數中進行載入前的處理，首先找出中間位置的xyz，並利用TransformFilter將模型移動到中間位置，以上動作完成後，會於Renderer中新增Actor，接著就可以看到剛才載入的模型了。

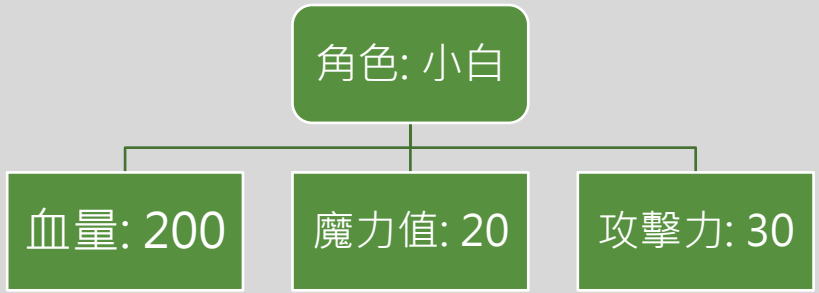
```
}
//預先處理vtkPolyData
vtkSmartPointer<vtkPolyData> ProcessingEngine::preprocessPolydata(const vtkSmartPointer<vtkPolyData> inputData) const
{
    // Center the polygon
    double center[3];
    inputData->GetCenter(center);

    vtkSmartPointer<vtkTransform> translation = vtkSmartPointer<vtkTransform>::New();
    //找出畫布的中間位置的XYZ
    translation->Translate(-center[0], -center[1], -center[2]);
    //將設定好的x,y,z丟到移動模型的函數內
    vtkSmartPointer<vtkTransformPolyDataFilter> transformFilter = vtkSmartPointer<vtkTransformPolyDataFilter>::New();
    //傳入模型數據
    transformFilter->SetInputData(inputData);
    transformFilter->SetTransform(translation);
    //更新資料
    transformFilter->Update();
```

41041220

Model – 模型類別

每個載入的模型，都會被定義為Model這個類別，需要用到的Filter、Mapper或Actor，都是在這裡做設定。






```
Model::Model(vtkSmartPointer<vtkPolyData> modelData)
    : m_modelData{modelData}
{
    // 將模型放於Z軸的零
    m_positionZ = -m_modelData->GetBounds()[4];

    vtkSmartPointer<vtkTransform> translation = vtkSmartPointer<vtkTransform>::New();
    //設定XYZ
    translation->Translate(m_positionX, m_positionY, m_positionZ);
    //將設定好的x,y,z丟到移動模型的函數內
    m_modelFilterTranslate = vtkSmartPointer<vtkTransformPolyDataFilter>::New();
    m_modelFilterTranslate->SetInputData(m_modelData);
    m_modelFilterTranslate->SetTransform(translation);
    //更新數據
    m_modelFilterTranslate->Update();

    //預設模型的顏色映射
    m_modelMapper = vtkSmartPointer<vtkPolyDataMapper>::New();
    m_modelMapper->SetInputConnection(m_modelFilterTranslate->GetOutputPort());
    //阻止顏色映射
    m_modelMapper->ScalarVisibilityOff();

    // 設定模型預設的著色法
    m_modelActor = vtkSmartPointer<vtkActor>::New();
    m_modelActor->SetMapper(m_modelMapper);
    m_modelActor->GetProperty()->SetInterpolationToFlat();

    //預設顏色及透明度
    m_modelActor->GetProperty()->SetAmbient(0.1);//光照
    m_modelActor->GetProperty()->SetDiffuse(0.7);//漫反射光
    m_modelActor->GetProperty()->SetSpecular(0.3);//镜反射光
    this->setColor(m_defaultModelColor);//設定的模型顏色
    //預設modelActor位置
    m_modelActor->SetPosition(0.0, 0.0, 0.0);
}
```

41041220

ProcessingEngine - 功能設定

在設定模型功能前，會先從m_models去取得每一個model，然後使用model的Actor進行功能設定。

```
}
//模型顯示
void ProcessingEngine::setModelsRepresentation(const int modelsRepresentationOption) const
{
    for (const std::shared_ptr<Model>& model : m_models)
    {
        model->getModelActor()->GetProperty()->SetRepresentation(modelsRepresentationOption);
    }
}
//設定模型透明度
void ProcessingEngine::setModelsOpacity(const double modelsOpacity) const
{
    for (const std::shared_ptr<Model>& model : m_models)
    {
        model->getModelActor()->GetProperty()->SetOpacity(modelsOpacity);
    }
}
//設定模型Gouraud插值
void ProcessingEngine::setModelsGouraudInterpolation(const bool enableGouraudInterpolation) const
{
    for (const std::shared_ptr<Model>& model : m_models)
    {   //如果啟用Gouraud的開關，就對模型進行Gouraud插值法
        if (enableGouraudInterpolation)
        {
            model->getModelActor()->GetProperty()->SetInterpolationToGouraud();
        }
        else
        {
            model->getModelActor()->GetProperty()->SetInterpolationToFlat();
        }
    }
}
//更新模型顏色
void ProcessingEngine::updateModelsColor() const
{
    for (const std::shared_ptr<Model>& model : m_models)
    {
        model->updateModelColor();
    }
}
```

41041222

QVTKFboItem

主要的兩個作用：

1. 作為Handler、Renderer、Command之間的橋樑
2. 刷新渲染器

當模型或視角改變時，會刷新介面

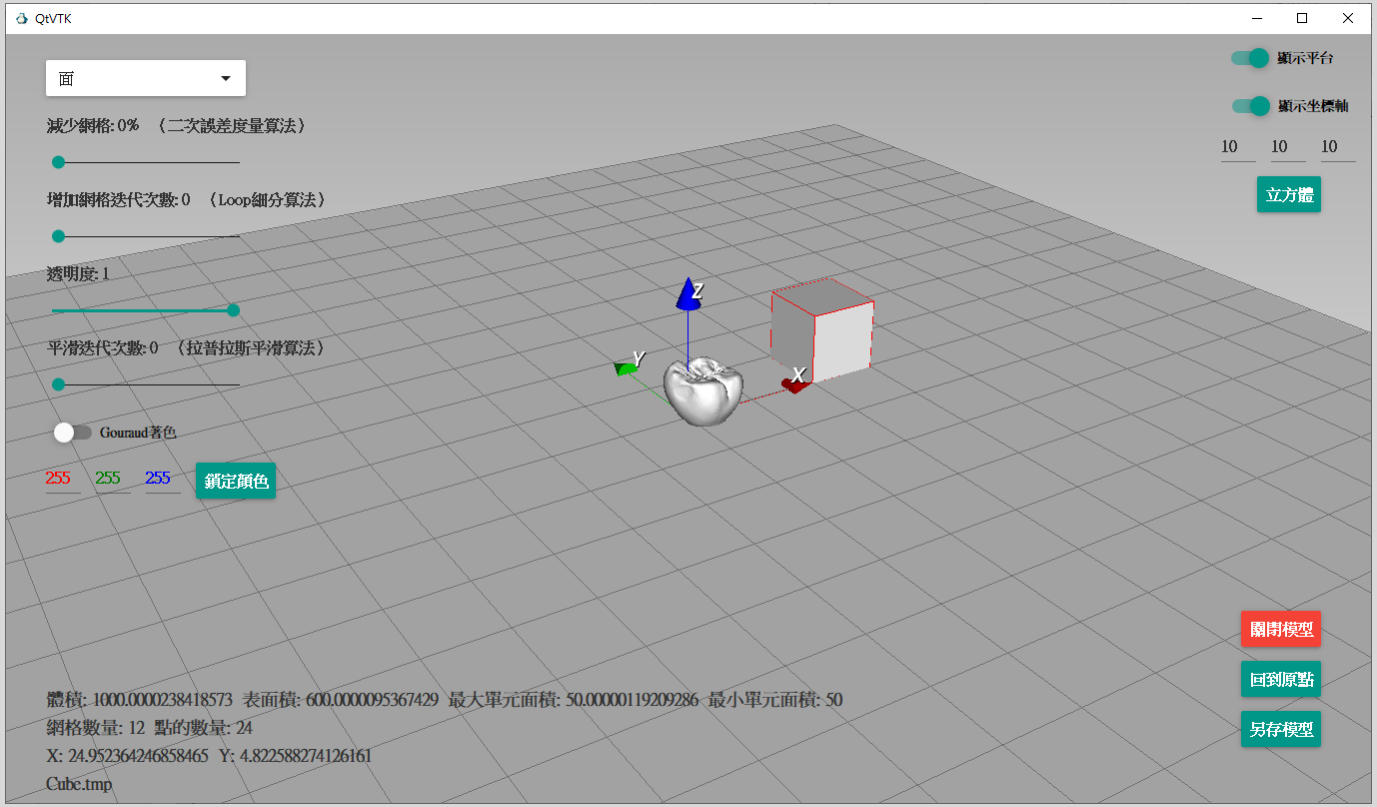
刷新同時會自動調用到ProcessingEngine的功能

```
260  void QVTKFramebufferObjectItem::addModelFromFile(const QUrl &modelPath)
261  {
262      qDebug() << "QVTKFramebufferObjectItem::addModelFromFile";
263
264      // 使用模型增加的Class，傳入參數(vtk渲染器, 處理引擎, 模型位置)
265      CommandModelAdd *command = new CommandModelAdd(m_vtkFboRenderer, m_processingEngine, modelPath);
266
267      // 建立連線，用於檢查模型是否加入完成
268      connect(command, &CommandModelAdd::ready, this, &QVTKFramebufferObjectItem::update);
269      connect(command, &CommandModelAdd::done, this, &QVTKFramebufferObjectItem::addModelFromFileDone);
270
271      command->start();
272
273      // 新增任務並執行
274      this->addCommand(command);
275  }
276
277  void QVTKFramebufferObjectItem::setModelsRepresentation(const int representationOption)
278  {
279      if (m_modelsRepresentationOption != representationOption)
280      {
281          m_modelsRepresentationOption = representationOption;
282          update(); // 刷新渲染器
283      }
284  }
285
286  double QVTKFramebufferObjectItem::getSelectedModelPositionX() const
287  {
288      // 取得選擇的模型座標X
289      return m_vtkFboRenderer->getSelectedModelPositionX();
290  }
291
292  double QVTKFramebufferObjectItem::getSelectedModelPositionY() const
293  {
294      // 取得選擇的模型座標Y
295      return m_vtkFboRenderer->getSelectedModelPositionY();
296  }
```

41041222

QVTKFboRenderer

1. 處理滑鼠事件(點擊/拖曳/滾輪)
2. 渲染介面(背景/平台/相機)
3. Pick事件



41041222

QVTKFboRenderer - 處理滑鼠事件

```
127     // Process mouse event
128     if (m_mouseEvent && !m_mouseEvent->isAccepted())
129     {
130         m_vtkRenderWindowInteractor->SetEventInformationFlipY(m_mouseEvent->x(), m_mouseEvent->y(),
131                                                               (m_mouseEvent->modifiers() & Qt::ControlModifier) > 0 ? 1 : 0,
132                                                               (m_mouseEvent->modifiers() & Qt::ShiftModifier) > 0 ? 1 : 0, 0,
133                                                               m_mouseEvent->type() == QEvent::MouseButtonDblClick ? 1 : 0);
134
135         if (m_mouseEvent->type() == QEvent::MouseButtonPress)
136         {
137             m_vtkRenderWindowInteractor->InvokeEvent(vtkCommand::LeftButtonPressEvent, m_mouseEvent.get());
138         }
139         else if (m_mouseEvent->type() == QEvent::MouseButtonRelease)
140         {
141             m_vtkRenderWindowInteractor->InvokeEvent(vtkCommand::LeftButtonReleaseEvent, m_mouseEvent.get());
142         }
143
144         m_mouseEvent->accept();
145     }
146
147     // Process move event
148     if (m_moveEvent && !m_moveEvent->isAccepted())
149     {
150         if (m_moveEvent->type() == QEvent::MouseMove && m_moveEvent->buttons() & Qt::RightButton)
151         {
152             m_vtkRenderWindowInteractor->SetEventInformationFlipY(m_moveEvent->x(), m_moveEvent->y(),
153                                                                   (m_moveEvent->modifiers() & Qt::ControlModifier) > 0 ? 1 : 0,
154                                                                   (m_moveEvent->modifiers() & Qt::ShiftModifier) > 0 ? 1 : 0, 0,
155                                                                   m_moveEvent->type() == QEvent::MouseButtonDblClick ? 1 : 0);
156
157             m_vtkRenderWindowInteractor->InvokeEvent(vtkCommand::MouseMoveEvent, m_moveEvent.get());
158         }
```

41041222

QVTKFboRenderer - 渲染介面

```
188     // Model transformations
189
190     CommandModel *command;
191     while (!m_vtkFboItem->isCommandsQueueEmpty())
192     {
193         m_vtkFboItem->lockCommandsQueueMutex();
194
195         command = m_vtkFboItem->getCommandsQueueFront();
196         if (!command->isReady())
197         {
198             m_vtkFboItem->unlockCommandsQueueMutex();
199             break;
200         }
201         m_vtkFboItem->commandsQueuePop();
202
203         m_vtkFboItem->unlockCommandsQueueMutex();
204
205         command->execute();
206     }
207
208     // Reset the view-up vector. This improves the interaction of the camera with the plate.
209     m_renderer->GetActiveCamera()->SetViewUp(0.0, 0.0, 1.0);
210
211     // Extra actions
212     m_processingEngine->setModelsRepresentation(m_modelsRepresentationOption);
213     m_processingEngine->setModelsOpacity(m_modelsOpacity);
214     m_processingEngine->setModelsGouraudInterpolation(m_modelsGouraudInterpolation);
215     m_processingEngine->updateModelsColor();
216
217     // Render
218     m_vtkRenderWindow->Render();
219     m_vtkRenderWindow->PopState();
220
221     m_vtkFboItem->window()->resetOpenGLState();
```

理解程式碼流程

41041222

QVTKFboRenderer -模型Pick事件

```
388  void QVTKFramebufferObjectRenderer::selectModel(const int16_t x, const int16_t y)
389  {
390      qDebug() << "QVTKFramebufferObjectRenderer::selectModel()";
391
392      // Compensate the y-axis flip for the picking
393      m_picker->Pick(x, m_renderer->GetSize()[1] - y, 0, m_renderer);
394
395      // Get pick position
396      double clickPosition[3];
397      m_picker->GetPickPosition(clickPosition);
398      m_clickPositionZ = clickPosition[2];
399
400      if (m_selectedActor == m_picker->GetActor())
401      {
402          if (m_selectedModel)
403          {
404              m_selectedModel->setMouseDeltaXY(clickPosition[0] - m_selectedModel->getPositionX(), clickPosition[1] - m_selectedModel->getPositionY());
405          }
406          return;
407      }
408
409      // Disconnect signals
410      if (m_selectedModel)
411      {
412          this->clearSelectedModel();
413      }
414
415      // Pick the new actor
416      m_selectedActor = m_picker->GetActor();
417
418      m_selectedModel = this->getSelectedModelNoLock();
419
420      if (m_selectedActor)
421      {
422          qDebug() << "QVTKFramebufferObjectRenderer::selectModel(): picked actor" << m_selectedActor;
423
424          m_selectedModel->setSelected(true);
425
```

改良、新增功能

41041223

功能用途

改良：

在研讀過作者提供的版本後，發現原程式碼的功能實現為採用一次性控制所有模型。我們為了讓使用者能清楚分辨模型功能的差異性，因此將功能實現的方法重寫並將其模組化；另外在載入模型時，原程式碼只接受單一模型的輸入，經過我們改良，不僅能同時將多個模型輸入到畫面上，還能夠支援不同類型（*.stl *.obj *.ply *.vtk *.vtp）的檔案進行輸入；最後是選擇模型，原版選擇模型時無法知道選擇的是哪一個，為了方便使用者觀察，於被選擇模型的外觀上增加紅色外框。

新增：

減少/增加網格、平滑、模型資訊、生成立方體、生成球體、模型回到原點

顯示平台、顯示坐標軸 (可開關)

關閉模型、合併模型、模型另存新檔。

- 模型相關
- 顯示相關
- 輸入輸出相關

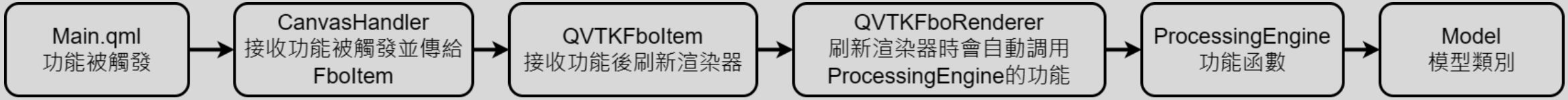
41041223

改良功能實現程式碼的框架結構

原有的框架結構是一個功能設定每一個模型，為了能識辨模型功能的差異性，因此決定重寫功能實現的方式，讓使用者可以控制單一模型的功能。我們將功能全部寫在Model類別，基於原程式碼中新增CommandModelActor用於跟Model中的功能對話，後續如果有需要新增功能，只要在Model中新增即可。

以下是功能實現的框架比較圖

原功能實現框架



改良後功能實現框架

如何新增功能？

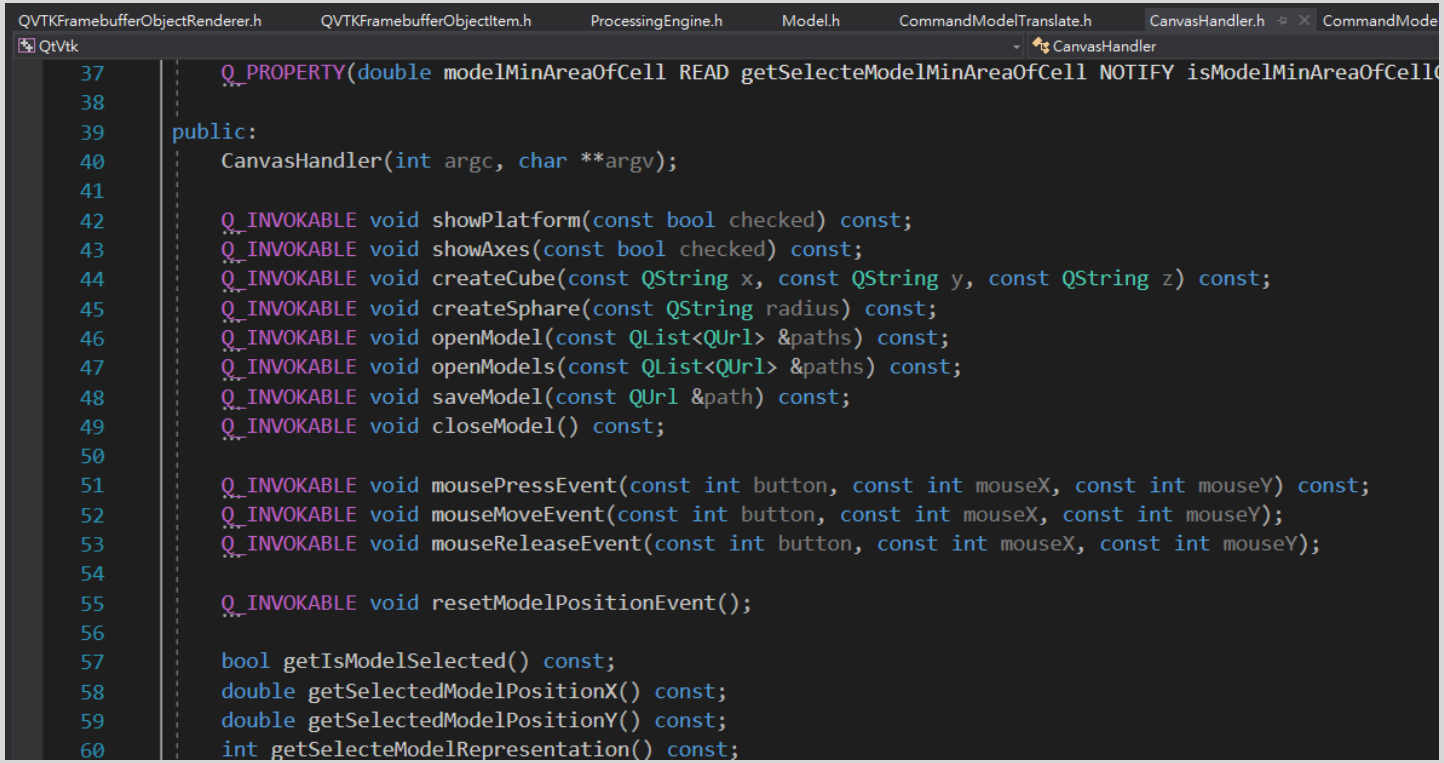
改良、新增功能

41041223

QML調用C++函數

在對接檔案(CanvasHanlder)中標頭檔的定義函數前面增加一個"Q_INVOKABLE"

Q_INVOKABLE是QML與C++交互的一種方式，加上該定義後，QML即可呼叫C++的函數



新增功能 - CanvasHandler

於CanvasHandler中，新增一個設定模型功能的函數

並使用"Q_INVOKABLE"將C++函數共享給qml使用

```
547  void CanvasHandler::setModelSmooth(const int value)
548  {
549      if (!m_vtkFboItem->isModelSelected())
550      {
551          // 沒有選擇模型就返回
552          return;
553      }
554      // 自定義的ActorParams Struct
555      // 一個結構體，用於模型功能的
556      // std::shared_ptr<Model> model; 指定的模型
557      // std::string mode; 功能類型
558      // int valueI{ 0 }; 整數數值
559      // double valueD{ 0 }; 雙精度浮點數值
560      // float valueF{ 0 }; 單精度浮點數值
561      // bool valueB = false; 布林值
562      // QColor color; 顏色數值
563      CommandModelActor::ActorParams_t actorParams;
564      actorParams.mode = "setModelSmooth";
565      actorParams.valueI = value;
566      m_vtkFboItem->actorModel(actorParams);
567  }
```

41041223

新增功能 - QVTKFboItem

於QVTKFboItem中，再次檢查模型是否被選擇，將actorData傳入任務的參數中並新增任務到佇列裡

addCommand函數會去調用刷新渲染器的函數(QVTKFboRenderer)

而刷新渲染器函數(QVTKFboRenderer)又會負責將佇列中的任務執行一遍。

```
275  void QVTKFramebufferObjectItem::actorModel(CommandModelActor::ActorParams_t & actorData)
276  {
277      // 檢查模型是否被選擇
278      if (actorData.model == nullptr)
279      {
280          // If no model selected yet, try to select one
281          actorData.model = m_vtkFboRenderer->getSelectedModel();
282
283          if (actorData.model == nullptr)
284          {
285              return;
286          }
287      }
288      // 新增任務 類型=Actor
289      this->addCommand(new CommandModelActor(m_vtkFboRenderer, actorData));
290  }
```

41041223

新增功能 - CommandModelActor

CommandModelActor中的setActor用於跟Model內的功能做對接，並設定使用者所選擇的功能。

```
16  void CommandModelActor::setActor()
17  {
18      if (m_actorParams.mode == "setModelRepresentation")
19      {
20          m_actorParams.model->setModelRepresentation(m_actorParams.valueI);
21      }
22      else if (m_actorParams.mode == "setModelDecreasePolygons")
23      {
24          m_actorParams.model->setModelDecreasePolygons(m_actorParams.valueD);
25      }
26      else if (m_actorParams.mode == "setModelIncreasePolygons")
27      {
28          m_actorParams.model->setModelIncreasePolygons(m_actorParams.valueI);
29      }
30      else if (m_actorParams.mode == "setModelOpacity")
31      {
32          m_actorParams.model->setModelOpacity(m_actorParams.valueD);
33      }
34      else if (m_actorParams.mode == "setModelGouraudInterpolation")
35      {
36          m_actorParams.model->setModelGouraudInterpolation(m_actorParams.valueB);
37      }
38      else if (m_actorParams.mode == "setModelColor")
39      {
40          m_actorParams.model->setModelColor(m_actorParams.color);
41      }
42      else if (m_actorParams.mode == "setModelSmooth")
43      {
44          m_actorParams.model->setModelSmooth(m_actorParams.valueI);
45      }
46  }
47
48  void CommandModelActor::execute()
49  {
50      this->setActor();
51  }
```

新增功能 - Model

最後在Model類中寫入功能函數

```
409     void Model::setModelSmooth(const int value)
410     {
411
412         if (value != m_Smooth)
413         {
414             m_Smooth = value;
415             m_modelSmoothFilter->SetNumberOfIterations(m_Smooth);
416             m_modelSmoothFilter->Update();
417             m_modelMapper->Update();
418             emit modelSmoothChanged(m_Smooth);
419         }
420     }
```

改良、新增功能

如何把數據顯示在畫面上？

改良、新增功能

41041223

C++傳輸到QML

利用Q_PROPERTY可以將變數傳輸到QML上， Q_PROPERTY必須繼承於Q_OBJECT類

Q_RROPERTY (類型, 屬性名稱, 方法(Read, Write, Member), 回調函數)

沒有使用到Read，可以使用Member來去使用Q_PROPERTY，後面為自己的變數

NOTIFY 後面是一個Signal回調函數，當值改變時，需要emit傳輸自己的信號

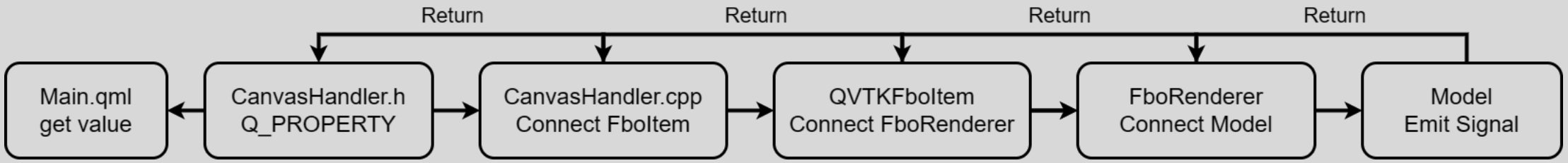
```
12  class CanvasHandler : public QObject
13  {
14      Q_OBJECT
15
16      Q_PROPERTY(bool showFileDialog MEMBER m_showFileDialog NOTIFY showFileDialogChanged)
17      Q_PROPERTY(bool showSaveFileDialog MEMBER m_showSaveFileDialog NOTIFY showFileDialogChanged)
18      Q_PROPERTY(bool showFilesDialog MEMBER m_showFilesDialog NOTIFY showFileDialogChanged)
19      Q_PROPERTY(bool isModelSelected READ getIsModelSelected NOTIFY isModelSelectedChanged)
20      Q_PROPERTY(double modelPositionX READ getSelectedModelPositionX NOTIFY selectedModelPositionXChanged)
21      Q_PROPERTY(double modelPositionY READ getSelectedModelPositionY NOTIFY selectedModelPositionYChanged)
22      Q_PROPERTY(int modelRepresentation READ getSelecteModelRepresentation NOTIFY isModelRepresentationChanged)
23      Q_PROPERTY(double modelDecreasePolygons READ getSelecteModelDecreasePolygons NOTIFY isModelDecreasePolygonsChanged)
24      Q_PROPERTY(int modelIncreasePolygons READ getSelecteModelIncreasePolygons NOTIFY isModelIncreasePolygonsChanged)
25      Q_PROPERTY(double modelOpacity READ getSelecteModelOpacity NOTIFY isModelOpacityChanged)
26      Q_PROPERTY(int modelSmooth READ getSelecteModelSmooth NOTIFY isModelSmoothChanged)
27      Q_PROPERTY(bool modelGouraud READ getSelecteModelGI NOTIFY isModelGIChanged)
28      Q_PROPERTY(int modelColorR READ getSelecteModelColorR NOTIFY isModelColorRChanged)
29      Q_PROPERTY(int modelColorG READ getSelecteModelColorG NOTIFY isModelColorGChanged)
30      Q_PROPERTY(int modelColorB READ getSelecteModelColorB NOTIFY isModelColorBChanged)
31      Q_PROPERTY(int modelPolygons READ getSelecteModelPolygons NOTIFY isModelPolygonsChanged)
32      Q_PROPERTY(int modelPoints READ getSelecteModelPoints NOTIFY isModelPointsChanged)
33      Q_PROPERTY(QString modelFileName READ getSelecteModelFileName NOTIFY isModelFileNameChanged)
34      Q_PROPERTY(double modelVolume READ getSelecteModelVolume NOTIFY isModelVolumeChanged)
35      Q_PROPERTY(double modelSurfaceArea READ getSelecteModelSurfaceArea NOTIFY isModelSurfaceAreaChanged)
36      Q_PROPERTY(double modelMaxAreaOfCell READ getSelecteModelMaxAreaOfCell NOTIFY isModelMaxAreaOfCellChanged)
37      Q_PROPERTY(double modelMinAreaOfCell READ getSelecteModelMinAreaOfCell NOTIFY isModelMinAreaOfCellChanged)
```

改良、新增功能

41041223

C++傳輸到QML

做好前面的宣告後，還需要進行連線，觀察原程式碼後發現，原作者的想法是將每一個檔案互相串連，最後交由Model emit信號，因此參考他的方法，會得到如下圖所示的流程

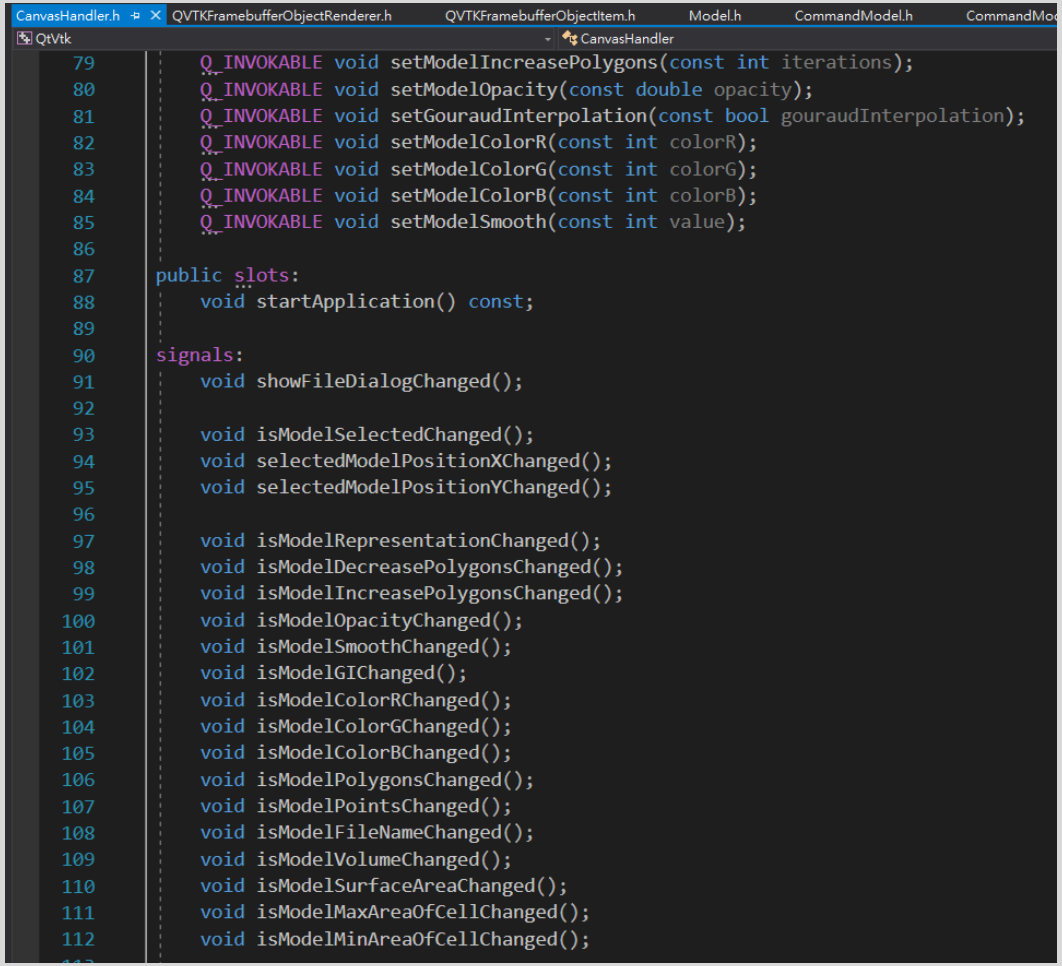


改良、新增功能

41041223

C++傳輸到QML

部分程式碼展示




```
79      Q_INVOKABLE void setModelIncreasePolygons(const int iterations);
80      Q_INVOKABLE void setModelOpacity(const double opacity);
81      Q_INVOKABLE void setGouraudInterpolation(const bool gouraudInterpolation);
82      Q_INVOKABLE void setModelColorR(const int colorR);
83      Q_INVOKABLE void setModelColorG(const int colorG);
84      Q_INVOKABLE void setModelColorB(const int colorB);
85      Q_INVOKABLE void setModelSmooth(const int value);
86
87  public slots:
88      void startApplication() const;
89
90  signals:
91      void showFileDialogChanged();
92
93      void isModelSelectedChanged();
94      void selectedModelPositionXChanged();
95      void selectedModelPositionYChanged();
96
97      void isModelRepresentationChanged();
98      void isModelDecreasePolygonsChanged();
99      void isModelIncreasePolygonsChanged();
100     void isModelOpacityChanged();
101     void isModelSmoothChanged();
102     void isModelGIChanged();
103     void isModelColorRChanged();
104     void isModelColorGChanged();
105     void isModelColorBChanged();
106     void isModelPolygonsChanged();
107     void isModelPointsChanged();
108     void isModelFileNameChanged();
109     void isModelVolumeChanged();
110     void isModelSurfaceAreaChanged();
111     void isModelMaxAreaOfCellChanged();
112     void isModelMinAreaOfCellChanged();
```

改良、新增功能

41041223

C++傳輸到QML

部分程式碼展示

QVTKFramebuffer...jectRenderer.cpp | QVTKFramebufferObjectItem.cpp | CanvasHandler.cpp | CommandModelTranslate.cpp | CommandModelActor.cpp | CommandModel.cpp | CommandModelSave.cpp | Model.cpp

QtVtk → CanvasHandler → setModelColorR(const int colorR)

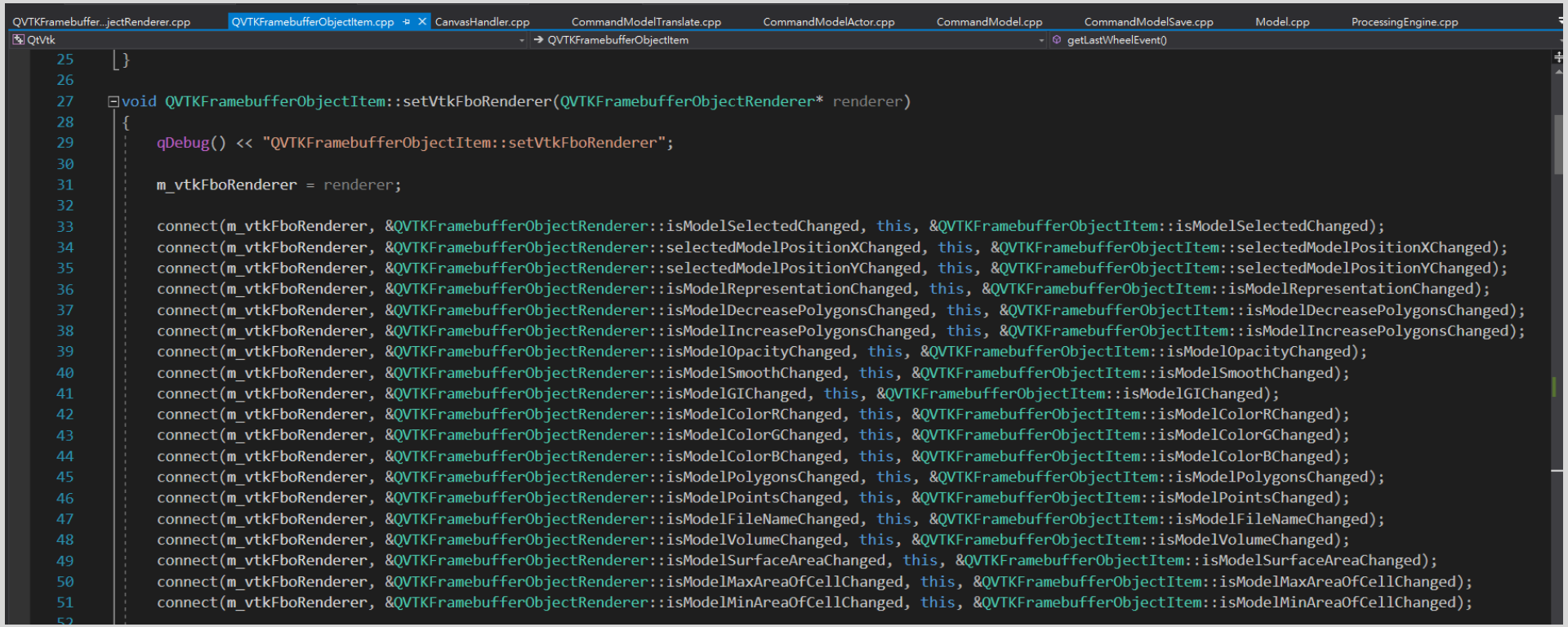
```
40      // We cannot use smart pointers because this object must be deleted by QML
41      QObject *rootObject = engine.rootObjects().first();
42      m_vtkFboItem = rootObject->findChild<QVTKFramebufferObjectItem*>("vtkFboItem");
43
44      // Give the vtkFboItem reference to the CanvasHandler
45      if (m_vtkFboItem)
46      {
47          qDebug() << "CanvasHandler::CanvasHandler: setting vtkFboItem to CanvasHandler";
48
49          m_vtkFboItem->setProcessingEngine(m_processingEngine);
50
51          connect(m_vtkFboItem, &QVTKFramebufferObjectItem::rendererInitialized, this, &CanvasHandler::startApplication);
52          connect(m_vtkFboItem, &QVTKFramebufferObjectItem::isModelSelectedChanged, this, &CanvasHandler::isModelSelectedChanged);
53          connect(m_vtkFboItem, &QVTKFramebufferObjectItem::selectedModelPositionXChanged, this, &CanvasHandler::selectedModelPositionXChanged);
54          connect(m_vtkFboItem, &QVTKFramebufferObjectItem::selectedModelPositionYChanged, this, &CanvasHandler::selectedModelPositionYChanged);
55          connect(m_vtkFboItem, &QVTKFramebufferObjectItem::isModelRepresentationChanged, this, &CanvasHandler::isModelRepresentationChanged);
56          connect(m_vtkFboItem, &QVTKFramebufferObjectItem::isModelDecreasePolygonsChanged, this, &CanvasHandler::isModelDecreasePolygonsChanged);
57          connect(m_vtkFboItem, &QVTKFramebufferObjectItem::isModelIncreasePolygonsChanged, this, &CanvasHandler::isModelIncreasePolygonsChanged);
58          connect(m_vtkFboItem, &QVTKFramebufferObjectItem::isModelOpacityChanged, this, &CanvasHandler::isModelOpacityChanged);
59          connect(m_vtkFboItem, &QVTKFramebufferObjectItem::isModelSmoothChanged, this, &CanvasHandler::isModelSmoothChanged);
60          connect(m_vtkFboItem, &QVTKFramebufferObjectItem::isModelGIChanged, this, &CanvasHandler::isModelGIChanged);
61          connect(m_vtkFboItem, &QVTKFramebufferObjectItem::isModelColorRChanged, this, &CanvasHandler::isModelColorRChanged);
62          connect(m_vtkFboItem, &QVTKFramebufferObjectItem::isModelColorGChanged, this, &CanvasHandler::isModelColorGChanged);
63          connect(m_vtkFboItem, &QVTKFramebufferObjectItem::isModelColorBChanged, this, &CanvasHandler::isModelColorBChanged);
64          connect(m_vtkFboItem, &QVTKFramebufferObjectItem::isModelPolygonsChanged, this, &CanvasHandler::isModelPolygonsChanged);
65          connect(m_vtkFboItem, &QVTKFramebufferObjectItem::isModelPointsChanged, this, &CanvasHandler::isModelPointsChanged);
66          connect(m_vtkFboItem, &QVTKFramebufferObjectItem::isModelFileNameChanged, this, &CanvasHandler::isModelFileNameChanged);
67          connect(m_vtkFboItem, &QVTKFramebufferObjectItem::isModelVolumeChanged, this, &CanvasHandler::isModelVolumeChanged);
68          connect(m_vtkFboItem, &QVTKFramebufferObjectItem::isModelSurfaceAreaChanged, this, &CanvasHandler::isModelSurfaceAreaChanged);
69          connect(m_vtkFboItem, &QVTKFramebufferObjectItem::isModelMaxAreaOfCellChanged, this, &CanvasHandler::isModelMaxAreaOfCellChanged);
70          connect(m_vtkFboItem, &QVTKFramebufferObjectItem::isModelMinAreaOfCellChanged, this, &CanvasHandler::isModelMinAreaOfCellChanged);
71      }
```

改良、新增功能

41041223

C++傳輸到QML

部分程式碼展示



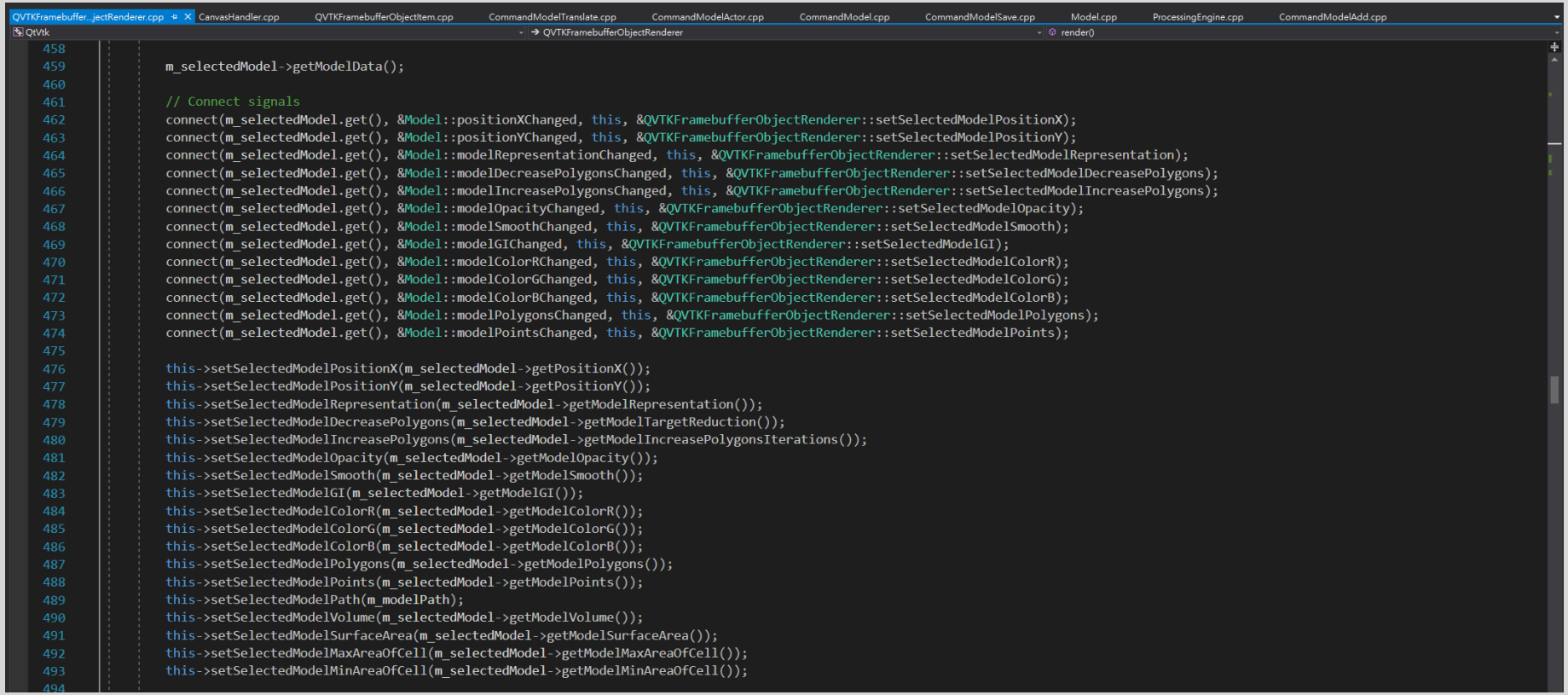

```
25      }
26
27      void QVTKFramebufferObjectItem::setVtkFboRenderer(QVTKFramebufferObjectRenderer* renderer)
28      {
29          qDebug() << "QVTKFramebufferObjectItem::setVtkFboRenderer";
30
31          m_vtkFboRenderer = renderer;
32
33          connect(m_vtkFboRenderer, &QVTKFramebufferObjectRenderer::isModelSelectedChanged, this, &QVTKFramebufferObjectItem::isModelSelectedChanged);
34          connect(m_vtkFboRenderer, &QVTKFramebufferObjectRenderer::selectedModelPositionXChanged, this, &QVTKFramebufferObjectItem::selectedModelPositionXChanged);
35          connect(m_vtkFboRenderer, &QVTKFramebufferObjectRenderer::selectedModelPositionYChanged, this, &QVTKFramebufferObjectItem::selectedModelPositionYChanged);
36          connect(m_vtkFboRenderer, &QVTKFramebufferObjectRenderer::isModelRepresentationChanged, this, &QVTKFramebufferObjectItem::isModelRepresentationChanged);
37          connect(m_vtkFboRenderer, &QVTKFramebufferObjectRenderer::isModelDecreasePolygonsChanged, this, &QVTKFramebufferObjectItem::isModelDecreasePolygonsChanged);
38          connect(m_vtkFboRenderer, &QVTKFramebufferObjectRenderer::isModelIncreasePolygonsChanged, this, &QVTKFramebufferObjectItem::isModelIncreasePolygonsChanged);
39          connect(m_vtkFboRenderer, &QVTKFramebufferObjectRenderer::isModelOpacityChanged, this, &QVTKFramebufferObjectItem::isModelOpacityChanged);
40          connect(m_vtkFboRenderer, &QVTKFramebufferObjectRenderer::isModelSmoothChanged, this, &QVTKFramebufferObjectItem::isModelSmoothChanged);
41          connect(m_vtkFboRenderer, &QVTKFramebufferObjectRenderer::isModelGIChanged, this, &QVTKFramebufferObjectItem::isModelGIChanged);
42          connect(m_vtkFboRenderer, &QVTKFramebufferObjectRenderer::isModelColorRChanged, this, &QVTKFramebufferObjectItem::isModelColorRChanged);
43          connect(m_vtkFboRenderer, &QVTKFramebufferObjectRenderer::isModelColorGChanged, this, &QVTKFramebufferObjectItem::isModelColorGChanged);
44          connect(m_vtkFboRenderer, &QVTKFramebufferObjectRenderer::isModelColorBChanged, this, &QVTKFramebufferObjectItem::isModelColorBChanged);
45          connect(m_vtkFboRenderer, &QVTKFramebufferObjectRenderer::isModelPolygonsChanged, this, &QVTKFramebufferObjectItem::isModelPolygonsChanged);
46          connect(m_vtkFboRenderer, &QVTKFramebufferObjectRenderer::isModelPointsChanged, this, &QVTKFramebufferObjectItem::isModelPointsChanged);
47          connect(m_vtkFboRenderer, &QVTKFramebufferObjectRenderer::isModelFileNameChanged, this, &QVTKFramebufferObjectItem::isModelFileNameChanged);
48          connect(m_vtkFboRenderer, &QVTKFramebufferObjectRenderer::isModelVolumeChanged, this, &QVTKFramebufferObjectItem::isModelVolumeChanged);
49          connect(m_vtkFboRenderer, &QVTKFramebufferObjectRenderer::isModelSurfaceAreaChanged, this, &QVTKFramebufferObjectItem::isModelSurfaceAreaChanged);
50          connect(m_vtkFboRenderer, &QVTKFramebufferObjectRenderer::isModelMaxAreaOfCellChanged, this, &QVTKFramebufferObjectItem::isModelMaxAreaOfCellChanged);
51          connect(m_vtkFboRenderer, &QVTKFramebufferObjectRenderer::isModelMinAreaOfCellChanged, this, &QVTKFramebufferObjectItem::isModelMinAreaOfCellChanged);
```

改良、新增功能

41041223

C++傳輸到QML

部分程式碼展示




```
458
459         m_selectedModel->getModelData();
460
461         // Connect signals
462         connect(m_selectedModel.get(), &Model::positionXChanged, this, &QVTKFramebufferObjectRenderer::setSelectedModelPositionX);
463         connect(m_selectedModel.get(), &Model::positionYChanged, this, &QVTKFramebufferObjectRenderer::setSelectedModelPositionY);
464         connect(m_selectedModel.get(), &Model::modelRepresentationChanged, this, &QVTKFramebufferObjectRenderer::setSelectedModelRepresentation);
465         connect(m_selectedModel.get(), &Model::modelDecreasePolygonsChanged, this, &QVTKFramebufferObjectRenderer::setSelectedModelDecreasePolygons);
466         connect(m_selectedModel.get(), &Model::modelIncreasePolygonsChanged, this, &QVTKFramebufferObjectRenderer::setSelectedModelIncreasePolygons);
467         connect(m_selectedModel.get(), &Model::modelOpacityChanged, this, &QVTKFramebufferObjectRenderer::setSelectedModelOpacity);
468         connect(m_selectedModel.get(), &Model::modelSmoothChanged, this, &QVTKFramebufferObjectRenderer::setSelectedModelSmooth);
469         connect(m_selectedModel.get(), &Model::modelGIChanged, this, &QVTKFramebufferObjectRenderer::setSelectedModelGI);
470         connect(m_selectedModel.get(), &Model::modelColorRChanged, this, &QVTKFramebufferObjectRenderer::setSelectedModelColorR);
471         connect(m_selectedModel.get(), &Model::modelColorGChanged, this, &QVTKFramebufferObjectRenderer::setSelectedModelColorG);
472         connect(m_selectedModel.get(), &Model::modelColorBChanged, this, &QVTKFramebufferObjectRenderer::setSelectedModelColorB);
473         connect(m_selectedModel.get(), &Model::modelPolygonsChanged, this, &QVTKFramebufferObjectRenderer::setSelectedModelPolygons);
474         connect(m_selectedModel.get(), &Model::modelPointsChanged, this, &QVTKFramebufferObjectRenderer::setSelectedModelPoints);
475
476         this->setSelectedModelPositionX(m_selectedModel->getPositionX());
477         this->setSelectedModelPositionY(m_selectedModel->getPositionY());
478         this->setSelectedModelRepresentation(m_selectedModel->getModelRepresentation());
479         this->setSelectedModelDecreasePolygons(m_selectedModel->getModelTargetReduction());
480         this->setSelectedModelIncreasePolygons(m_selectedModel->getModelIncreasePolygonsIterations());
481         this->setSelectedModelOpacity(m_selectedModel->getModelOpacity());
482         this->setSelectedModelSmooth(m_selectedModel->getModelSmooth());
483         this->setSelectedModelGI(m_selectedModel->getModelGI());
484         this->setSelectedModelColorR(m_selectedModel->getModelColorR());
485         this->setSelectedModelColorG(m_selectedModel->getModelColorG());
486         this->setSelectedModelColorB(m_selectedModel->getModelColorB());
487         this->setSelectedModelPolygons(m_selectedModel->getModelPolygons());
488         this->setSelectedModelPoints(m_selectedModel->getModelPoints());
489         this->setSelectedModelPath(m_modelPath);
490         this->setSelectedModelVolume(m_selectedModel->getModelVolume());
491         this->setSelectedModelSurfaceArea(m_selectedModel->getModelSurfaceArea());
492         this->setSelectedModelMaxAreaOfCell(m_selectedModel->getModelMaxAreaOfCell());
493         this->setSelectedModelMinAreaOfCell(m_selectedModel->getModelMinAreaOfCell());
494
```

C++傳輸到QML

部分程式碼展示

QVTKFramebuffer...jectRenderer.cpp | Model.cpp | CanvasHandler.cpp | QVTKFramebufferObjectItem.cpp | CommandModelTranslate.cpp | CommandModelActor.cpp

QtVtk → QVTKFramebufferObjectRenderer

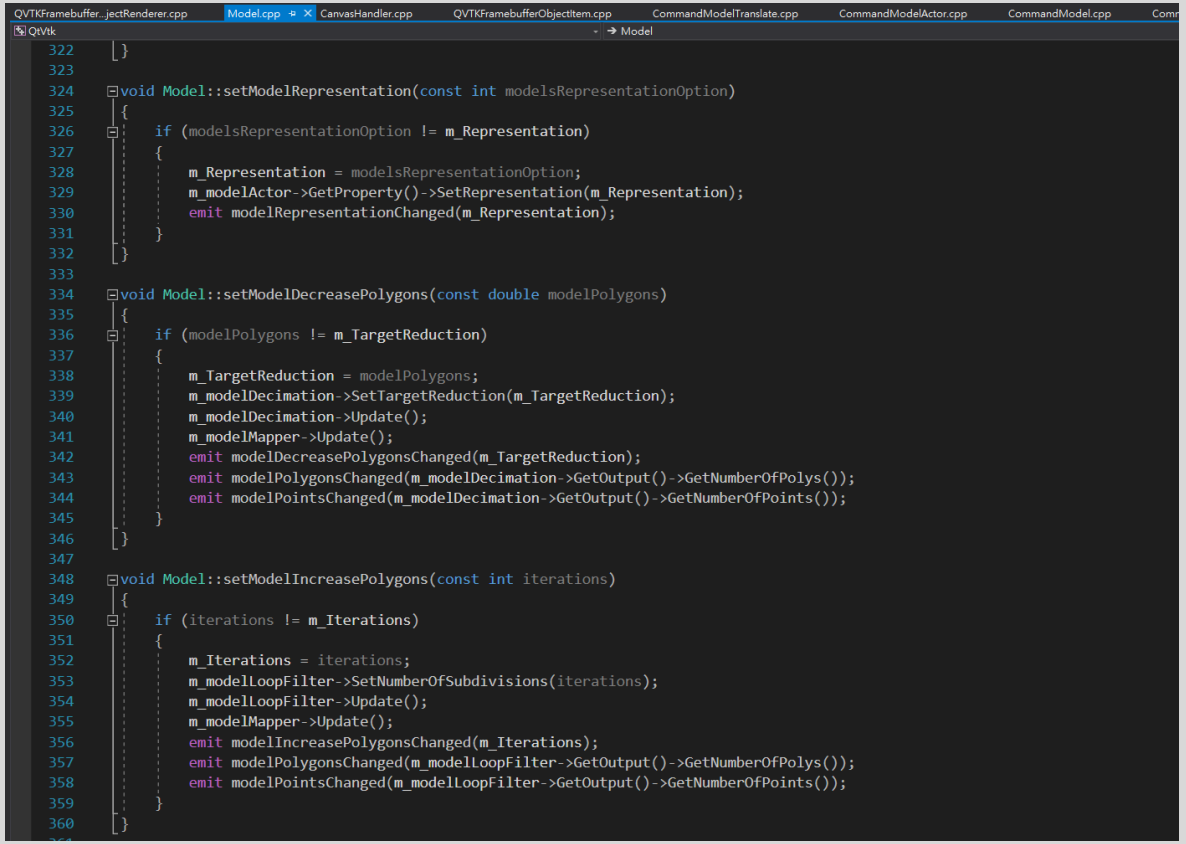
```
719  void QVTKFramebufferObjectRenderer::setSelectedModelRepresentation(const int option)
720  {
721      if (m_modelRepresentation != option)
722      {
723          m_modelRepresentation = option;
724          emit isModelRepresentationChanged();
725      }
726  }
727
728  void QVTKFramebufferObjectRenderer::setSelectedModelDecreasePolygons(const double polygons)
729  {
730      if (m_modelDecreasePolygons != polygons)
731      {
732          m_modelDecreasePolygons = polygons;
733          emit isModelDecreasePolygonsChanged();
734      }
735  }
736
737  void QVTKFramebufferObjectRenderer::setSelectedModelIncreasePolygons(const int iterations)
738  {
739      if (m_modelIncreasePolygons != iterations)
740      {
741          m_modelIncreasePolygons = iterations;
742          emit isModelIncreasePolygonsChanged();
743      }
744  }
745
746  void QVTKFramebufferObjectRenderer::setSelectedModelOpacity(const double opacity)
747  {
748      if (m_modelOpacity != opacity)
749      {
750          m_modelOpacity = opacity;
751          emit isModelOpacityChanged();
752      }
753  }
```

改良、新增功能

41041223

C++傳輸到QML

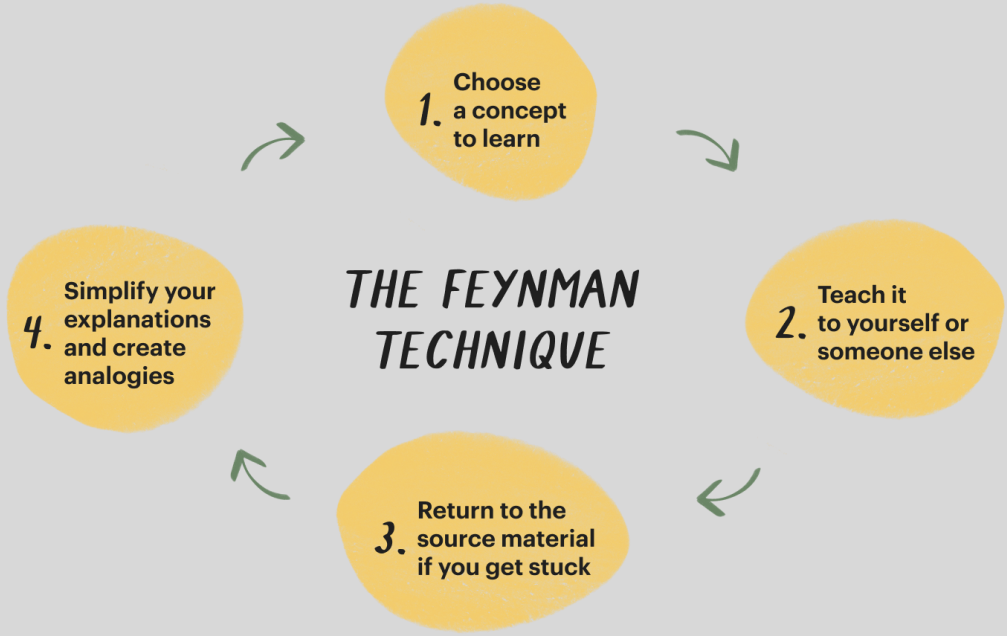
部分程式碼展示




```cpp
}

void Model::setModelRepresentation(const int modelsRepresentationOption)
{
    if (modelsRepresentationOption != m_Representation)
    {
        m_Representation = modelsRepresentationOption;
        m_modelActor->GetProperty()->SetRepresentation(m_Representation);
        emit modelRepresentationChanged(m_Representation);
    }
}

void Model::setModelDecreasePolygons(const double modelPolygons)
{
    if (modelPolygons != m_TargetReduction)
    {
        m_TargetReduction = modelPolygons;
        m_modelDecimation->SetTargetReduction(m_TargetReduction);
        m_modelDecimation->Update();
        m_modelMapper->Update();
        emit modelDecreasePolygonsChanged(m_TargetReduction);
        emit modelPolygonsChanged(m_modelDecimation->GetOutput()->GetNumberOfPolys());
        emit modelPointsChanged(m_modelDecimation->GetOutput()->GetNumberOfPoints());
    }
}

void Model::setModelIncreasePolygons(const int iterations)
{
    if (iterations != m_Iterations)
    {
        m_Iterations = iterations;
        m_modelLoopFilter->SetNumberOfSubdivisions(iterations);
        m_modelLoopFilter->Update();
        m_modelMapper->Update();
        emit modelIncreasePolygonsChanged(m_Iterations);
        emit modelPolygonsChanged(m_modelLoopFilter->GetOutput()->GetNumberOfPolys());
        emit modelPointsChanged(m_modelLoopFilter->GetOutput()->GetNumberOfPoints());
    }
}
```

總結

本次的資料結構期末報告，我們研究了VTK及Qt的應用並加以實作。比起在課堂上聽課，透過手動實作與小組討論的方式，才能讓知識牢記在腦中。能夠讀懂別人的程式碼並修改是一件很重要的事，未來在業界工作時，或許會需要接手前輩留下的程式碼，因此我認為這次的學習經驗是非常有幫助的。



未來展望

- 目前功能方面稍有欠佳，並沒有到非常完整，無法達到產品的標準，但對程式方面已經足夠了解並實現模組化，想要增加功能方面只是時間上的問題。
- 在測試時已經能夠取得使用者滑鼠點擊的位置，可藉此延伸更加實用的功能，如:
  對模型進行任意變形、選取網格、融合、切割、測量距離等
  可惜的是時間不允許，無法做進一步嘗試，只能止步於此。

Demo展示、提問時間

Thanks for watching